Making the Impossible Possible

Christina Boura, Virginie Lallemand, María Naya-Plasencia, Valentin Suder

Making the Impossible Possible

Christina Boura[1], Virginie Lallemand[2], María Naya-Plasencia[2], Valentin Suder[3]

[1] UVSQ, France
christina.boura@uvsq.fr
[2] Inria, France
virginie.lallemand@inria.fr, maria.naya_plasencia@inria.fr
[3] University of Waterloo
valentin@suder.xyz

Abstract. This paper introduces new techniques and correct complexity analyses for impossible differential cryptanalysis, a powerful block cipher attack. We show how the key schedule of a cipher impacts an impossible differential attack and we provide a new formula for the time complexity analysis that takes this parameter into account. Further, we show, for the first time, that the technique of multiple differentials can be applied to impossible differential attacks. Then, we demonstrate how this technique can be combined in practice with multiple impossible differentials or with the so-called state-test technique. To support our proposal, we implemented the above techniques on small-scale ciphers and verified their efficiency and accuracy in practice. We apply our techniques to the cryptanalysis of ciphers including AES-128, CRYPTON-128, ARIA-128, CLEFIA-128, Camellia-256 and LBlock. All of our attacks significantly improve previous impossible differential attacks and generally achieve the best memory complexity among all previous attacks against these ciphers.

Keywords. block ciphers, impossible differential attacks, multiple differentials, key schedule, implementations, AES, CRYPTON, ARIA, CLEFIA, Camellia, LBlock.

1 Introduction

Impossible differential cryptanalysis is a very powerful attack against block ciphers introduced independently by Knudsen [18] and Biham et al. [3]. The idea of these attacks is to exploit impossible differentials, which are differentials occurring with probability zero. The general approach is then to extend the impossible differential by some rounds, possibly in both directions, guess the key bits that intervene in these rounds and check whether a trial pair is partially encrypted (or decrypted) to the impossible differential. In this case, we know that the guessed key bits are certainly wrong and we can remove the subsequent key from the candidate key space. Impossible differential attacks have been successfully applied to a large variety of block ciphers, based both on the SPN and the Feistel construction. In some cases, they yield the best cryptanalysis against the targeted cipher; this is the case for the standardized Feistel cipher

Camellia [25, 10], for example. Furthermore, impossible differential attacks were for a long time the most successful attacks against AES-128 [39, 27, 29].

Recently, a generic complexity analysis of impossible differential attacks against Feistel ciphers was presented [10]. Thanks to this generalized vision, several flaws in previous attacks were detected and many new attacks were proposed. Our work is the natural extension of the analysis given in [10] that inspired since its publication new results and analyses (e.g. [11, 4, 32, 38, 5, 23]). The techniques introduced in this paper correct, complete, and improve the techniques and analyses given in [10]. We further show how to combine all of these concepts in practice to mount optimized impossible differential attacks. In our applications, and in contrast to [10], we consider SPN ciphers. It is important to recall here that the time complexity formula of [10] is a lower-bound approximation. This approximation is most of the times met in practice, but as shown in [11], some counter-examples may exist. So, as already pointed out in [10], we insist here on the fact that the exact complexity of each attack needs to be carefully computed.

1.1 Our contributions

The main contributions of this paper.

Correction of the time complexity approximation taking into account the role of the key schedule. The first contribution of this paper is related to the role that the nature of the key schedule plays in an impossible differential attack. Indeed, if the key schedule is non-linear and has sufficiently good diffusion, then it is usually not trivial to translate guessed information on a subkey into information on the master key. In this case, the key schedule can be seen as a black box between the first and last subkeys. We show that this implies a new term must be taken into account in the time complexity evaluation. This remark results in a more accurate estimate of the time complexity, and then leads to a correction of the time complexity formula provided in [10].

New technique for improving data complexity: Multiple differentials. **(Not to confuse with the technique of multiple impossible differentials introduced in [27]).** Our second contribution is to apply the technique of multiple differentials to impossible differential attacks, in order to reduce the data complexity. While this idea seems quite natural, these two techniques had never been combined before. Applying this idea, sometimes in combination with multiple impossible differentials, leads to improved attacks against many ciphers as we prove through some concrete applications.

Experimental verifications of the introduced techniques. Our third contribution is to experimentally verify the theoretical complexities of our techniques and those of [10]. More precisely, we have implemented the state-test technique and the use of multiple (impossible) differentials with toy examples. In the state-test case, we show that the estimated complexity gain matches the real gain. With respect

to the multiple (impossible) differentials, we have performed several experiments, leading to the following important conclusions:

- When the wanted probability of keeping a random partial key as candidate is around 1/2 (implying a certain needed number of pairs), the use of any multiple output (impossible) differential will lead to a data complexity matching the formulas. In the case of multiple input (impossible) differentials, the obtained complexities will match the theoretical complexities only if the amount and form of needed pairs allow to optimally exploit the plaintext structures, and will be slightly reduced otherwise.
- When the wanted probability of keeping a random key is much smaller, e.g. if we only want to keep the correct secret key at the end of the attack, the corresponding amount of pairs will be slightly increased if multiple (impossible) differentials are considered (whether they are input or output ones), as a direct consequence of the higher number of key bits being involved. This previously unknown side effect, will also imply a divergence with respect to the formulas. This divergence can be summed up to the previous one in the case of non optimal input configurations.

To the best of our knowledge, this is the first time these techniques have been implemented.

Multiple impossible differentials vs. simple impossible differentials. We provide a discussion on the comparison of an attack that exploits multiple impossible differentials with an attack with similar parameters but that exploits only one impossible differential. An interesting question that arises in this type of situations is whether there are cases where an attack using a single impossible differential provides better complexities than an attack exploiting multiple impossible differentials. To answer this question, we provide in Section 5 an application against the block cipher ARIA-128 and we demonstrate that while the data complexity is always worse in the single case, the time complexity of an attack with a single impossible differential can sometimes be slightly better.

Application to various block ciphers. We apply our techniques to a variety of block ciphers. Our goal is to demonstrate the practical combination of our techniques with some of those of [10] (such as the state-test technique, for example). This is a technical task which was not correctly treated in [10]. Table 1 shows the complexities of all of our attacks together with a summary of the best known cryptanalyses on the targeted ciphers. As the table shows, the techniques of this paper permitted us to improve the attacks of [10] against the Feistel ciphers LBlock, CLEFIA-128 and Camellia-256 (without FL layers and whitening keys). In fact, most of the attacks that we provide improve on the memory complexities of the best known attacks. We also improve on the best known impossible differential attacks against three SPN block ciphers, namely AES-128, CRYPTON-128, and ARIA-128. While other types of cryptanalysis have led to more powerful attacks on these three ciphers, our techniques still yield an interesting improvement on previous impossible differential attacks. Each of these applications illustrates

a different combination of our methods. Only our application against 7-round AES-128 will be treated in full here (the other applications are sketched more briefly). This attack has the best memory complexity among all known attacks against AES-128 (though its time complexity does not improve on the best). This attack gives a perfect illustration of how to practically combine almost all of the techniques introduced in this paper.

Algorithm	**Rounds**	**Data** (**CP**)	**Time**	**Memory** (**Blocks**)	**Technique**	**Ref.**
AES-128 [14]	7	$2^{106.2}$	$2^{110.2}$	$2^{90.2}$	ID	[29]
	7	2^{105}	$2^{105} + 2^{99}$	2^{90}	MITM	[13]
	7	2^{97}	2^{99}	2^{98}	MITM	[13]
	7	2^{121}	$2^{121} + 2^{83}$	2^{74}	MITM §	[12]
	7	2^{113}	$2^{113} + 2^{75}$	2^{82}	MITM §	[12]
	7	$2^{113.1}$	$2^{113.1} + 2^{105.1}$	$2^{74.1}$	ID	**Sec. 4**
	7	2^{105}	$2^{106.88}$	2^{74}	ID	**Sec. 4**
CRYPTON-128 [24]	7	2^{97}	$2^{97.2}$	2^{100}	Trunc. Diff.	[17]
	7	2^{121}	$2^{121} + 2^{116.2}$	2^{119} †	ID	[30]
	7	$2^{114.92}$	$2^{114.92} + 2^{113.7}$	$2^{88.5}$	ID	**Sec. 4**
	8	2^{126}	$2^{126.2}$	2^{100}	Trunc. Diff.	[17]
ARIA-128 [19]	6	2^{113}	$2^{121.6}$	2^{113*}	ID	[21]
	6	2^{121}	$2^{121} + 2^{112}$	2^{121*}	ID	[37]
	6	$2^{120.5}$	$2^{120.5} + 2^{104.5}$	2^{121*}	ID	[21]
	6	2^{120}	$2^{120} + 2^{96}$	2^{120*}	ID	[22]
	6	2^{111}	$2^{111} + 2^{82}$	2^{71}	ID	**Sec. 4**
	7	$2^{105.8}$	$2^{105.8} + 2^{100.99}$	$2^{79.73}$	LC	[26]
CLEFIA-128 [34]	13	$2^{111.02}$	$2^{122.26}$	$2^{82.6}$	ID ⋄	[10]
	13	$2^{114.58}$	$2^{116.16}$	$2^{83.16}$	ID ⋄	[10]
	13	$2^{114.4}$	$2^{114.4}$	2^{80}	ID	**Sec. 4**
	13	2^{99}	2^{99}	2^{80}	Trunc. Diff.	[20]
	14	2^{100}	2^{108}	$2^{101.3}$	Trunc. Diff.	[20]
Camellia-256‡ [1]	14	2^{120}	$2^{250.5}$	2^{120}	ID	[25]
	14	2^{118}	2^{220}	2^{173}	ID ⋄	[10]
	14	$2^{117.7}$	$2^{215.7}$	$2^{166.7}$	ID	**Sec. 4**
LBlock [36]	23	2^{59}	$2^{75.36}$	2^{74}	ID	[10]
	23	$2^{63.87}$	$2^{74.30}$	2^{60}	ZC	[6]
	23	$2^{55.5}$	2^{72}	2^{65}	ID	**Sec. 4**

Table 1. Summary of best single-key attacks against AES-128, CRYPTON-128, ARIA-128, CLEFIA-128, Camellia-256‡ and LBlock. * Estimated memory requirements since not given in the original papers. ⋄ Incorrect result not taking into account the key-schedule. † Complexity estimated in [28]. ‡ Without whitening keys and FL layers. § Additional trade-offs of the attacks in [12] provided by P. Derbez (private communication).

.

The rest of the paper is organized as follows. Section 2 presents our new techniques and remarks on impossible differential attacks. The role of the key schedule is discussed, the combination of multiple differentials and impossible multiple differentials is presented and a corrected formula for estimating the time complexity of an attack is given. Section 3 is dedicated to the implementation of the introduced techniques on toy ciphers. Finally, Section 4 presents our attacks against AES-128, CRYPTON-128 and ARIA-128.

2 Impossible differential cryptanalysis

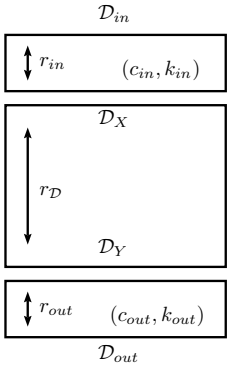
We provide here the basic principles of an impossible differential attack, and introduce the notation that will be used throughout this paper.

2.1 An overview of impossible differential cryptanalysis

We start by recalling the framework introduced in [10].

An impossible differential attack against an n-bit block cipher, parametrized by a key $\mathcal{K}$ of length K, starts with the discovery of an impossible differential composed of an input difference $\mathcal{D}_X$ that propagates after $r_{\mathcal{D}}$ rounds to an output difference $\mathcal{D}_Y$ with probability zero. After this, one extends this differential r_{in} rounds backwards to obtain a difference that we will denote $\mathcal{D}_{in}$ and r_{out} rounds forwards to obtain a difference called $\mathcal{D}_{out}$. The $\log_2$ of the size of a set $\mathcal{D}$ will be denoted by Δ.

The two appended differentials are used to eliminate the candidate keys that encrypt and decrypt data to the impossible differential. Indeed, if for a candidate key both differentials $\mathcal{D}_{in} \to \mathcal{D}_X$ and $\mathcal{D}_{out} \to \mathcal{D}_Y$ are satisfied, then this key is certainly wrong as it leads to an impossible differential and must therefore be rejected.



Two important quantities in an impossible differential attack are the total number of key bits that intervene in the appended rounds and the number of bit-conditions that must be satisfied in order to get $\mathcal{D}_X$ from $\mathcal{D}_{in}$ and $\mathcal{D}_Y$ from $\mathcal{D}_{out}$. We will therefore let k_{in} (resp. k_{out}) denote the number of key bits that have to be guessed during the first (resp. last) rounds, and $|k_{in} \cup k_{out}|$ the entropy of the involved key bits when considering relations due to the key schedule. Similarly, c_{in} (resp. c_{out}) will denote the number of bit-conditions to be verified during the first (resp. last) rounds.

We continue by briefly reminding the way to determine the number of pairs needed for the attack.

The probability that for a given key, a pair of inputs already satisfying the differences $\mathcal{D}_{in}$ and $\mathcal{D}_{out}$ verifies all the $(c_{in} + c_{out})$ bit-conditions is $2^{(c_{in}+c_{out})}$. In other words, this is the probability that for a pair of inputs satisfying the difference $\mathcal{D}_{in}$ and whose outputs satisfy the difference $\mathcal{D}_{out}$, a key from the

possible key set is discarded. Therefore, by repeating the procedure with N different input (or output) pairs, the probability that a trial key is kept in the candidate keys set is

$$P = (1 - 2^{-(c_{in}+c_{out})})^N.$$

There is not a unique strategy for choosing the amount of input (or output) pairs N. This choice principally depends on the overall time complexity, which is influenced by N, and the induced data complexity. Different trade-offs are therefore possible. A popular strategy, generally used by default is to choose N such that only the right key is left after the sieving procedure. This amounts to choose P as

$$P = (1 - 2^{-(c_{in}+c_{out})})^N < \frac{1}{2^{|k_{in} \cup k_{out}|}}.$$

However, as shown in [10], a different approach can be applied helping to reduce the number of pairs needed for the attack and to offer better trade-offs between the data and time complexity. More precisely, it is permitted to consider smaller values of N. By proceeding like this, one will be probably left with more than one key in the candidate keys set and will need to proceed to an exhaustive search among the remaining candidates, but the total time complexity of the attack will probably be much lower. In practice, one will start by considering values of N such that P is slightly smaller than $\frac{1}{2}$ so to reduce the exhaustive search by at least one bit. So N should be chosen such as

$$P = (1 - 2^{-(c_{in}+c_{out})})^N \approx e^{-N \times 2^{-(c_{in}+c_{out})}} < \frac{1}{2}. \tag{1}$$

We remind here that the quantity N determines the memory complexity of the attack.

The data complexity of an attack can be determined by the following formula given in [10].

$$C_N = \max \left\{ \min_{\Delta \in \{\Delta_{in}, \Delta_{out}\}} \left\{ \sqrt{N 2^{n+1-\Delta}} \right\}, N 2^{n+1-\Delta_{in}-\Delta_{out}} \right\}, \tag{2}$$

where Δ_{in} is the number of active bits in $\mathcal{D}_{in}$ ($\log_2$ of the dimension of the input space) and Δ_{out} is the number of active bits in $\mathcal{D}_{out}$.

Finally, we remind the analysis of the time complexity presented in [10]. We recall again, that the formula provided is a lower-bound approximation of the time complexity. This is due to the fact that each of the terms of this formula represents the minimum complexity of the operations that should be done in order to accomplish each step.

By following the early abort technique, the attack consists in storing the N pairs and testing out step by step the key candidates, by reducing at each time the size of the remaining possible pairs. The time complexity is then determined by three quantities. The first term is the cost C_N, that is the amount of needed data (see Formula (2)) for obtaining the N pairs, where N is such that $P < 1/2$. The second term corresponds to the number of candidate keys $2^{|k_{in} \cup k_{out}|}$, multiplied by the average cost of testing the remaining pairs. For all

the applications that we have studied, this cost can be very closely approximated by $\left(N + 2^{|k_{in} \cup k_{out}|} \frac{N}{2^{c_{in}+c_{out}}}\right) C'_E$, where C'_E is the ratio of the cost of partial encryption to the full encryption. Finally, the third term is the cost of the exhaustive search for the key candidates still in the candidate keys set after the sieving. By taking into account the cost of one encryption C_E, the approximation of the time complexity is given by

$$C_T = \left(C_N + \left(N + 2^{|k_{in} \cup k_{out}|} \frac{N}{2^{c_{in}+c_{out}}}\right) C'_E + 2^K P\right) C_E. \tag{3}$$

Obviously, as the attack complexity should be smaller than that of exhaustive search, the quantity C_T should be smaller than $2^K C_E$. In Section 2.5, after discussing the role of the key schedule in an impossible differential attack and after presenting our new techniques, we provide a corrected time complexity formula that takes all of the above into account.

In all of the applications that we provide at the end of this paper, we aim to derive different possible trade-offs for the time, data and memory complexity of an attack. For this reason, we introduce a parameter ε offering this possibility. More precisely, we take $N = 2^{c_{in}+c_{out}+\varepsilon}$. The data and time complexity formulas are subsequently modified. Different values of ε provide different complexity trade-offs.

In [10], it was said that $\mathcal{D}_{in}$ and $\mathcal{D}_{out}$ were obtained by allowing the differences $\mathcal{D}_X$ and $\mathcal{D}_Y$ to propagate with probability 1 in the backward and forward directions respectively. However, we point out here that this restriction is not necessary. In the case of Feistel constructions it is a common technique to propagate the $\mathcal{D}_X$ and $\mathcal{D}_Y$ differences with probability 1, as one usually does not have the choice of doing this in a different manner. However, in the case of SPN ciphers using AES-type matrices for diffusion, considering probabilistic propagation clearly makes sense as it considerably increases the number of possibilities for extending the impossible differential, and therefore offers more flexibility to the attacker for finding the best parameters for the cryptanalysis. If we take for example the case of AES, there are usually many possibilities for extending an active state after the `MixColumns` operation. An attacker can thus choose among all these possible cases and take the transitions that provide the best parameters for her attack.

This remark has important consequences for the data complexity of some attacks. Indeed, as seen by the formulas given in [10], if we allow only transitions $\mathcal{D}_X \to \mathcal{D}_{in}$ and $\mathcal{D}_Y \to \mathcal{D}_{out}$ of probability 1, then the equalities $\Delta_{in} - c_{in} = \Delta_X$ and $\Delta_{out} - c_{out} = \Delta_Y$ are true by Bayes' Theorem. Thus, the minimal data complexity, given by C_N, is in this case $2^{n+1-\Delta_X-\Delta_Y}$, meaning for example that if only one impossible differential is considered, the attack on some ciphers will not work, without the use of any special techniques, because of a lack of data. This is the case for impossible differential attacks against the block cipher SIMON, for example, where $\Delta_X = \Delta_Y = 0$. Indeed, as can be seen in [10], both $\mathcal{D}_X$ and $\mathcal{D}_Y$ in the attack against SIMON have only one bit active, therefore the $\log_2$ of both these quantities is zero. This then leads to $C_N \geq 2^{n+1}$, implying

that the attack does not work. If the probability for choosing the input and output differences is not 1, then this does not hold anymore and more flexibility is available for choosing the different trade-off parameters.

2.2 On the key schedule seen as a black box

The first contribution of this paper is to reveal that the nature of the key schedule has an impact on the complexity of an impossible differential attack. Indeed, if the cipher's key schedule is strongly non-linear, the first few subkeys have necessarily a very complicated relation with the subkeys of the last rounds. Note that the link between the nature of the key-schedule and the complexity of the underlying attack has been independently reported by Derbez [11].

In the context of impossible differential attacks, in general one has to guess key bits that belong to subkeys that have a gap of some rounds between them. If the key schedule is complex, then it is not possible to directly translate the information guessed on the subkey bits into the same amount of information on the master key. For this reason, one has to complete the missing bits to some of the partially known subkeys of the first or of the last rounds until we have enough bits to compute through the key schedule (or its inverse). Once this done, one can verify if this way of completing the missing bits was correct by checking if the result matches with the previously known key bits of the subkeys found on the other side of the impossible differential.

Usually, the part of the key schedule that connects the subkeys of the first rounds to the subkeys of the last rounds can be seen as a black box, and the computation above should be taken into account in the estimation of the time complexity. Before providing the new term that has to be taken into account in such a situation, we briefly define a classification of the key schedules and give the resulting key bit guessing techniques that should be adopted. We further introduce two new notations, k_A and k_B, which permit us to partition the key bits to be guessed into two separate groups according to the three following cases:

Linear or almost linear key-schedules. In such a case, it is possible to directly translate the k_{in} and k_{out} bits of the first and last rounds in the same number of bits of the master key by using the key schedule. Therefore, we set $k_A = |k_{in} \cup k_{out}|$ and $k_B = 0$. For example, the block cipher LBlock has a key schedule of this type, and this was exploited in the attack provided in [9].

Complex key schedule of AES-type. In cases where it is very complicated to connect the k_{in} bits of the first rounds to the k_{out} bits of the last rounds, we simply set $k_A = k_{in}$ and $k_B = k_{out}$. The block ciphers AES, CRYPTON and ARIA belong to this group.

Complex key schedule of MISTY1 or Camellia-type. This category also includes ciphers with highly non-linear key schedules, however the partition of the key bits into first and last round bits is not always relevant. For example, Camellia-128's key schedule can be seen as dividing subkeys into two groups, where on the one hand the relation between subkeys of the same group is very easy to compute, but on the other hand it is very complicated to connect subkeys of different groups. The difference with the previous type of key schedule is that these two

groups do not exactly correspond to k_{in} and k_{out}. Therefore, in such a case k_A will represent the subkey bits of one group, while k_B the subkey bits of the other group. The block cipher CLEFIA has also a key schedule of this type.

We are now ready to introduce the term taking into consideration the black box phenomenon that has to be added to Eq. (3): $\min(2^{K-k_A}, 2^{K-k_B}) \cdot P \cdot 2^{k_A+k_B} \cdot C_{KS}$, where C_{KS} is the key schedule cost. The quantities $K-k_A$ and $K-k_B$ correspond to the number of missing key bits that have to be completed. The above term can be simply rewritten as

$$\min(2^{K+k_A}, 2^{K+k_B}) \cdot P \cdot C_{KS}.$$

This term, multiplied by $\max(2^{-k_A}, 2^{-k_B}) \cdot \frac{1}{C_{KS}}$, gives the number of candidate keys to test.

To conclude this paragraph, we emphasize that the remark presented in this section had never been pointed out before, and was not taken into consideration in [10]. Indeed, in many previous attacks, even if the key schedule of the analyzed cipher becomes highly nonlinear through the rounds, it was wrongly supposed that one guessed word of a subkey could directly be seen as one guessed word of the master key[4]. Of course, the classification given above does not take into account every possible key schedule that one can imagine. One can think of key schedules not fitting any of the above categories. However, in concrete constructions, we have not encountered such cases and we believe that the key schedules used in practice lie in one of the above classes.

2.3 Multiple differentials in impossible differential cryptanalysis

Multiple differential cryptanalysis [31] is a generalization of differential cryptanalysis in which several input and output differences are considered simultaneously. We show here that this technique can be successfully combined with impossible differential cryptanalysis to reduce the data complexity of an attack. To the best of our knowledge, the idea of combining these two techniques had never been considered before. Furthermore, as we demonstrate in the next section, this method can also be combined with multiple impossible differentials [35, 10] to further reduce the amount of data that an attacker requires.

The idea here is to consider several input differences $\mathcal{D}_{in}$ and several output differences $\mathcal{D}_{out}$, all of them corresponding to the same pair of differences $(\mathcal{D}_X, \mathcal{D}_Y)$ as depicted in Figure 1. This method recalls the idea from [16], where multiple differentials were applied to rebound-type distinguishers.

Considering multiple differentials provides the attacker with more input/output differences $\mathcal{D}_{in}, \mathcal{D}_{out}$, meaning that there are more choices for the input/output patterns of a pair. Indeed, in a chosen-plaintext attack, a plaintext pair will be

[4]Obviously, we can imagine key schedules where a subkey does not necessarily provide any additional information on subsequent ones, and in this case our formula cannot be applied in the state. However, we have not encountered such type of key schedule in none of our various applications, as we can imagine that such key schedules have bad implementation properties.

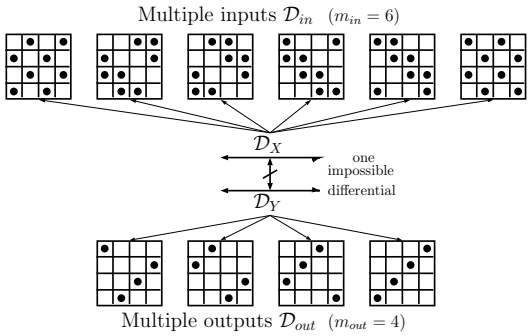


Fig. 1. Multiple inputs and multiple outputs

kept if the truncated difference of the corresponding ciphertexts is among the multiple output differences $\mathcal{D}_{out}$, leading to more choices than in an attack with a single $\mathcal{D}_{out}$. As we realized during our experiments, the input case is a bit more complicated to deal with, but globally it can be seen in the same way. Therefore, less data is needed to construct the pairs for the attack; this is the reason why this method helps to reduce the overall data complexity.

We define here two new variables, m_{in} and m_{out}, corresponding to the number of input/output multiple differentials taken into account for a single impossible differential. Following the same reasoning as in [10] where the data complexity of an attack using multiple impossible differentials was given as a function of the data complexity, C_N of a standard attack with the same parameters, we deduce that the new data complexity $C_{N'}$ is

$$C_{N'} = \frac{C_N}{m_{in} m_{out}}. \tag{4}$$

We show in the sequel how to correctly deal with the situation where different sets of key bits are related to the different $\mathcal{D}_{in}, \mathcal{D}_{out}$ differences. However, we note here that our analysis only considers multiple input and output differences of equal Hamming weight. Otherwise, the individual complexities might be non-equivalent, and in that case the differential with the highest complexity becomes the leading term without therefore improving the final complexity.

2.4 Multiple differentials with multiple impossible differentials

The idea of multiple impossible differentials, first introduced by Tsunoo et al. [35] and later formalized in [10], is to simultaneously consider several impossible differentials $(\mathcal{D}_X, \mathcal{D}_Y)$. This technique reduces the data complexity of the attack compared to a cryptanalysis that only exploits one impossible differential. This is due to the fact that using multiple impossible differentials implies less bit-conditions to be verified (as one has more choice), and the number of bit-conditions directly affects the number of pairs N and thus the amount of data, as can be seen in Eq. (2).

We introduce the idea of using multiple differentials and multiple impossible differentials together to further reduce the amount of data. If n_{in} is the number of input differences $\mathcal{D}_X$ and n_{out} the number of output differences $\mathcal{D}_{out}$, then the reduced data complexity by combining both techniques is

$$C'_N = \frac{C_N}{n_{in} n_{out} m_{in} m_{out}}. \tag{5}$$

This formula is directly derived from Eq. (4) and from the formula for the data complexity given in [10] for multiple impossible differentials.

In practice, whether the different differentials come from several impossible differentials $(\mathcal{D}_X, \mathcal{D}_Y)$ or from several input and/or output differences $(\mathcal{D}_{in}, \mathcal{D}_{out})$ will not change the way the complexity of the attack is affected, and we treat both types equally. For simplicity, we use *multiples* to refer to both multiple impossible differentials and multiple differentials. Applications of the above are provided in Section 4.

2.5 Putting it all together.

Multiples and black-box key-schedules. When considering several multiples, there can be different patterns for the groups of key bits of size k_A or k_B involved in the attack. In some cases these groups will be disjoint but this will not always be so. For the sake of simplicity, we concentrate on the case of output multiples (the analysis of input multiples is similar). We let $M = n_{in} n_{out} m_{in} m_{out}$ denote the total number of multiples. We let k_B^{inv} denote the number of k_B bits that are involved in at least one of these differentials (i.e. the union of all the sets of k_B bits), and $k_B^{int} = M \times k_B - k_B^{inv}$ the total number of redundant bits from k_B when we suppose that all the key bits are affected from all the differentials at the same time. In this case, the term to take the black-box phenomenon into account is:

$$2^{K-k_B^{inv}} \cdot (P^{1/M} \cdot 2^{k_A+k_B})^M \cdot 2^{-k_B^{int}} \cdot 2^{-k_A^{int}} \cdot C_{KS} = 2^K \cdot P \cdot 2^{k_A^{inv}} \cdot C_{KS}. \tag{6}$$

This previous term, multiplied by $2^{-k_A^{inv}} \cdot \frac{1}{C_{KS}}$, gives the number of candidate keys to test, while the last term of the complexity stays $2^K \cdot P \cdot C_E$. We omit the min here, since we can choose the roles of k_A and k_B. Several applications of this situation are provided in Section 4.

Given these formulas, the combination of both the state-test and the multiple (impossible) differentials is now straightforward. Combining everything, the new time complexity formula that we propose is

$$C_T = \left(C_N + \left(N + 2^{k_A+k_B} \frac{N}{2^{c_{in}+c_{out}}} \right) C'_E + 2^K \cdot P \cdot 2^{k_A^{inv}} \cdot C'_{KS} + 2^K \cdot P \right) C_E, \tag{7}$$

where C'_{KS} is the ratio of the cost of the key schedule compared to the full encryption. Our application against AES-128 gives an illustration of such a combination.

Multiples and state-test. The aim of the state-test technique, introduced in [10], is to eliminate some candidate keys without having to consider all of the possibilities for the involved key bits. This can be done, for example, by considering the value x of a word of size s of the internal part of the state needed to verify if a condition is satisfied in the second round. Typically, with a constant c from the diffusion layer and an invertible Sbox S, we would have $x = x' + cS(P_i + K_i) + K_j$, where x' is an already known value that we have computed with the knowledge of the plaintexts/ciphertexts and the already guessed key bits. The s-bit variable P_i, corresponds to the fixed part of the state, i.e. it has the same value for all the considered pairs. The variables K_i and K_j correspond to the not yet guessed nor determined involved parts of the key, of size s each. We easily see that if instead of guessing both variables K_i and K_j we directly guess the value $x + x'$, then we can perform the rest of the attack in a similar way, with a complexity reduced by s bits, as the number of guesses is reduced by this amount. Each guess of $x + x'$ will imply a disjoint set of possibilities for K_i and K_j, and considering all the values of $x' + x$ will provide all possible combinations of K_i and K_j. The attack is performed as before, where now we will determine the candidate values for $x + x'$. Note again that this is only possible because the value of P_i is fixed. This simplified version of the state-test technique combined with a simplified vision of multiple (impossible) differentials eases their combination. Consider a simple attack, i.e. implying a single impossible differential, performed with N_s number of pairs. Let P_s be the proportion of candidate keys that we obtain, and let C_{N_s} be the data complexity of the corresponding attack. The number of remaining key candidates is $2^{|k_{in} \cup k_{out}|} \cdot P_s 2^{K - |k_{in} \cup k_{out}|} = 2^K \cdot P_s$.

Now, suppose that we repeat this attack T times in parallel for different sets of data, possibly involving different key bits. While the parameters of the repeated attacks are the same as for the first one, the number of candidate keys left[5] will be $(2^{|k_{in} \cup k_{out}|} \cdot P_s)^T \cdot 2^{-k_{int}}$, where k_{int} is the total number of redundant bits from $\mathcal{K}$ when we consider all the key bits affected by all the multiple differentials together. The data complexity in this case is $T \cdot C_{N_s}$, for a proportion of keys ${P_s}^T$, and the time complexity is about $T \cdot C_{T_s}$. It is easy to see that when we perform a multiple instead of a parallel repetition we are following a similar procedure, but we can reuse the data. Therefore the data complexity of this multiple attack will be smaller, while the time and memory complexities will a priori stay the same.

Combining the above representations of the the state-test and multiple impossible differentials techniques, together with the new formula that correctly takes into account the key schedule when using multiple differentials, is now straightforward. The attack against AES-128 gives a detailed illustration of how these two methods can be combined.

[5] See the previous section for a more complete description when the key schedule is seen as a black Sbox.

3 Verification of the improvement techniques

In this section we detail the implementation experiments that we performed in order to verify the improvement techniques introduced both in this paper and in [10]. To the best of our knowledge, this is the first time that the state-test technique and the multiple (impossible) differential techniques have been implemented, and therefore validated. We emphasize the importance of implementation as the only means of corroborating the theoretical approaches.

Multiple differentials.

The scope of the first implementation experiment is to get a clear idea of the accuracy of the equations given in Section 2, and in particular Formula (5), when working with multiple (impossible) differentials.[6] For doing so, we considered a toy cipher corresponding to a 6-round Feistel network using blocks of $n = 32$ bits and whose round function has an SP structure. This round function is therefore composed of 3 operations: a bitwise key addition, the parallel application of a 4-bit Sbox (same as for PRESENT [8]) and finally the application of a MDS linear transformation P (same as for LED [15]).

For the sake of simplicity we considered independent round keys. We used a 4-round impossible differential, with an input difference of the form $\mathcal{D}_X = (0,0,0,0|a,0,0,0)$ (with a a non-null nibble) and an output difference $\mathcal{D}_Y = (x,y,z,0|0,0,0,0)$, where x, y and z are nibbles free of conditions[7]. We add one round before and after this differential and end up with the following parameters: $\mathcal{D}_{in} = (a,0,0,0|P(b,0,0,0))$ with a and b nibbles free of conditions, leading to $\Delta_{in} = 8$, $c_{in} = 4$ and $\mathcal{D}_{out} = (P(u,v,w,0)|(x,y,z,0)))$ (all nibbles free of conditions) leading to $\Delta_{out} = 24$, $c_{out} = 12$.

To start with, we consider a simple attack against the above toy cipher exploiting only one impossible differential and we suppose that our goal is to discard half of the candidate keys after the attack. According to Eq. (1), we need N pairs satisfying both $\mathcal{D}_{in}$ and $\mathcal{D}_{out}$, where N is such that

$$P = (1 - 2^{-(c_{in}+c_{out})})^N = (1 - 2^{-16})^N < \tfrac{1}{2}.$$

This leads to $N > 2^{15.47}$. We verified experimentally that the above formula is accurate by launching 10 tests on our toy cipher. Indeed, the experiment showed that we need in average $2^{23.44}$ pairs satisfying $\mathcal{D}_{in}$ to eliminate half of the candidate keys. This means that we have $2^{23.44-n+\Delta_{out}} = 2^{23.44-8} = 2^{15.44}$ pairs satisfying both $\mathcal{D}_{in}$ and $\mathcal{D}_{out}$. In the sequel of our implementation experiments

[6] When implementing an attack that takes advantage of multiple differentials, one fatally needs to append many rounds to the impossible differential in order to get a significant number of differentials. This leads then to a significant increase in the complexity of the implemented attack. For this reason, we have only considered in our experiments multiple impossible differentials. However, due to the similarity of these two notions, we conjecture that the conclusions of this section can be applied to both types of differentials.

[7] The impossibility of this differential comes from the MDS property of P.

we are interested in the evolution of the number of necessary pairs N when more than one impossible differential is used.

Furthermore, in the following experiments, we verify the accuracy of Formula (5) in the case of input multiples, in the case of output multiples, and in the case that the probability of keeping a key is not one half, but is equal to the inverse of the number of possible (involved) keys.

We first describe the simplest experiments, where we considered a probability of not discarding a key being 1/2, i.e., where half of the possible keys are eliminated after the attack.

Using multiple outputs. We are interested here in the evolution of the quantity of required pairs if we use multiple impossible differentials for the second half of the differential, i.e. if we exploit the 4 possible patterns (of same Hamming weight) for $\mathcal{D}_Y : \mathcal{D}_Y = (x, y, z, 0|0, 0, 0, 0)$, $(x, 0, y, z|0, 0, 0, 0)$, $(x, y, 0, z|0, 0, 0, 0)$ and $(x, y, z, 0|0, 0, 0, 0)$ (see Figure 2).

In such a case, if we have N pairs satisfying both $\mathcal{D}_{in}$ and one out of the 4 possible $\mathcal{D}_{out}$, the probability to not discard a key is not modified, as the conditions remain unchanged for all the 4 output possibilities, and is equal to $P = (1 - 2^{-16})^N$, which indicates that if we want to divide by 2 the number of possible keys, the required amount of pairs satisfying both $\mathcal{D}_{in}$ and (one of the) $\mathcal{D}_{out}$ is unchanged. On the other hand, since more output differences are valid, we need to encrypt less pairs satisfying $\mathcal{D}_{in}$ to find N pairs. Indeed, if n_{out} output differences are considered, we need to encrypt only a fraction of n_{out}^{-1} pairs satisfying $\mathcal{D}_{in}$ (see Figure 2).

The results of our experiments are given in Table 2. One can remark that these results correspond to what was predicted by theory.

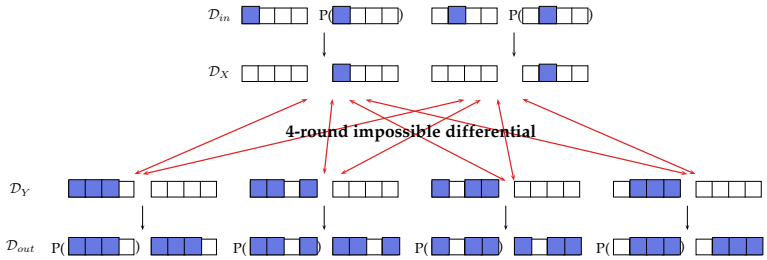
Table 2. Necessary amount of pairs N satisfying $\mathcal{D}_{in}$ in order to eliminate half of the candidate keys (average on 10 tests) and associated C_N. To decrease the data complexity, we use structures of size 2^8 that cover all possible values for a and b.

number of considered $\mathcal{D}_{out}$	1	2	3	4
theoretical value of $\log_2(N) + n - \Delta_{out}$	23.5	22.5	21.9	21.5
experimental value of $\log_2(N) + n - \Delta_{out}$	23.4	22.4	21.8	21.6
theoretical value of $\log_2(C_N)$	16.5	15.5	14.9	14.5
experimental value of $\log_2(C_N)$	16.5	15.4	14.8	14.6

Using multiple inputs and combination with multiple outputs. We consider here the case of several input differentials $\mathcal{D}_{in}$ and as we will see the situation is now slightly different. The probability of eliminating a key remains unchanged, so we still require the same amount of pairs N satisfying (one of the) $\mathcal{D}_{in}$ and $\mathcal{D}_{out}$.

The experiments we did on our toy cipher meet this theory. More precisely, we generated random pairs, alternatively following the first and the second $\mathcal{D}_{in}$,

Fig. 2. Attack configuration with 2 $\mathcal{D}_{in}$ and 4 $\mathcal{D}_{out}$.



and counted how many pairs are necessary to divide the set of possible keys by 2. It resulted that we need (average on 10 tests) $2^{22.5}$ pairs in each $\mathcal{D}_{in}$ if we use a single $\mathcal{D}_{out}$. This quantity decreases to $2^{21.5}$, $2^{21.0}$ and $2^{20.6}$ respectively for 2, 3 and 4 $\mathcal{D}_{out}$.

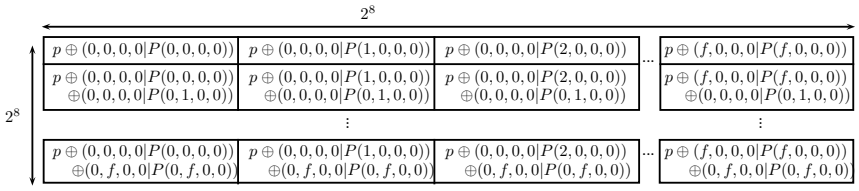
The gain from the multiple inputs would come from the fact that we can create pairs following one of the $\mathcal{D}_{in}$ in a clever way, by carefully selecting the plaintexts we encrypt. For instance, if we use 2 different $\mathcal{D}_{in}$, a nice choice would be to choose a random plaintext p and to encrypt the 2^{16} messages given by:

$$\{p \oplus (a,0,0,0|P(b,0,0,0)) \oplus (0,c,0,0|P(0,d,0,0)), a,b,c,d \in GF(2^4)\}$$

With such a set, we are able to make $2^{15} \times 2^8 = 2^{23}$ pairs satisfying each of the entering pattern, i.e. 2^{24} pairs satisfying one of the $\mathcal{D}_{in}$, while this amount of encryption would have given only 2^{23} pairs if only one $\mathcal{D}_{in}$ was exploited.

We can visualize such a structure as a 2-dimensional array: each line is made of 2^8 plaintexts that form a structure for the first $\mathcal{D}_{in}$, and each column makes a structure for the second $\mathcal{D}_{in}$ (see Figure 3).

Fig. 3. Efficient Structure to exploit two $\mathcal{D}_{in}$.



If we require a multiple of 2^{24} pairs satisfying one of the $\mathcal{D}_{in}$, the data gain should be of one half. However, if we require less pairs, building structures is not that obvious and the real gain could be smaller. If we consider 2 possible $\mathcal{D}_{in}$, a solution would be to create a structure similar to the one in Figure 3 with $2^{\ell_1} \leq 2^8$ lines and $2^{\ell_2} \leq 2^8$ columns, which would allow to build approximately $2^{\ell_1+2\ell_2-1}$ pairs satisfying the first $\mathcal{D}_{in}$ and $2^{\ell_2+2\ell_1-1}$ pairs satisfying the second

one. The aim is then to be able to build the needed amount of pairs satisfying one of the $\mathcal{D}_{in}$ ($2^{\ell_1+2\ell_2-1} + 2^{\ell_2+2\ell_1-1}$) while minimizing the number of encryptions ($2^{\ell_1+\ell_2}$). We have therefore verified that the given equations cannot always be met with respect to the multiples considered in the input, and the loss with respect to this will depend on the best way of building the structures. Some results show that, in the generic case with n_{in} differentials, the best configuration for having the smallest loss is to take maximal values for the first ℓ_i while not exceeding the needed N, and then complete the next one with the needed amount, while considering 1 for the others. This implies that using all the structures associated with a 1 won't improve the data complexity, as it will be useless, and the best possible improvement is achieved when considering as many different $\mathcal{D}_{in}$ as the various ℓ_i different from 1 that we have.

Choosing N in order to keep only the correct key. The situation becomes quite different if we are interested in keeping only the correct key. In this case, for a simple attack with one $\mathcal{D}_{in}$ and one $\mathcal{D}_{out}$, Eq. (1) becomes

$$P = (1 - 2^{-(c_{in}+c_{out})})^N < \frac{1}{2^{16}},$$

since 4 nibbles of the key intervene in the attack. This theoretic formula gives that N has to be bigger than $2^{19.47}$. We launched 10 experiments and were able to confirm this quantity in the non-multiple case; the average obtained is of $2^{27.5}$ pairs satisfying $\mathcal{D}_{in}$. When considering several possible $\mathcal{D}_{out}$, the number of possible involved key bits is going to be slightly increased, but this increase is enough to affect the data complexity. This was not taken into account in [10] nor in our theoretical formulas, and should be kept in mind. As can be seen in Table 3, this has a small but clear effect in the data needs, allowing to gain slightly less than what predicted.

Table 3. Necessary amount of pairs following the unique $\mathcal{D}_{in}$ in order to keep only the right key (average on 10 tests) and associated C_N. To decrease the data complexity, we use structures of size 2^8 that cover all the values for a and b.

number of considered $\mathcal{D}_{out}$	1	2	3	4
experimental value of $\log_2(N) + n - \Delta_{out}$	27.5	26.9	26.4	26.0
experimental value of $\log_2(C_N)$	20.5	19.9	19.4	19.0

For the sake of completeness, we have considered the case where the probability of not discarding a key is as small as necessary to only keep one key, and we consider at the same time 2 different $\mathcal{D}_{in}$ and up to 4 different $\mathcal{D}_{out}$. We see in Table 4 how the side effect of a higher number of involved key bits is a bit stronger here than in Table 3, as with two $\mathcal{D}_{in}$, this number is increased. We do not see here the effect of the non-optimal structures constructed with the $\mathcal{D}_{in}$, as in this particular case, the amount needed is big enough to optimally exploit such

structures. Therefore, the obtained experimental values for C_N nearly match the theoretical ones.

Table 4. Necessary amount of pairs satisfying one of the two $\mathcal{D}_{in}$ in order to keep only the right key (average on 10 tests) and associated C_N. In this case, taking complete structures allows to reduce the necessary amount of encryptions, as predicted by theory.

number of considered $\mathcal{D}_{out}$	1	2	3	4
experimental value of $\log_2(N) + n - \Delta_{out}$	27.9	27.2	26.7	26.3
experimental value of $\log_2(C_N)$	20	19.2	18.8	18.4

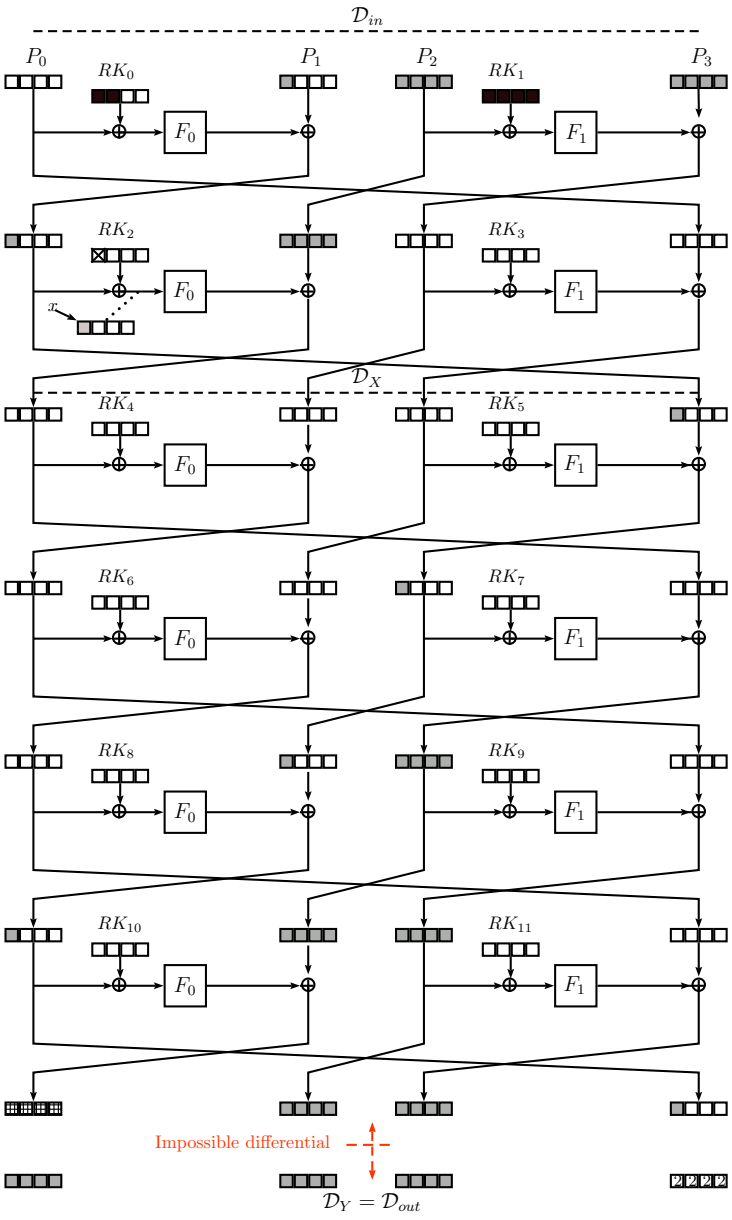
State-test technique. We describe here the experiments we performed to validate the state-test technique. For this purpose we used a slightly modified version of CLEFIA [34] in which we drop the word-size from 8 bits to 4 bits, adapting the internal functions to fit this new size. As depicted in Figure 4, we attack 6 rounds of such a CLEFIA with a 4-round impossible differential and the same 2-round input differential used in the attacks of [10].

For practical reasons, we suppose that the subkey RK_1 has already been guessed. We then applied the state-test technique on the value of the nibble denoted by x in Figure 4 to recover one nibble of the subkey RK_0 and one nibble of RK_2. We performed this experiment for 2^3 randomly chosen keys, using different amounts of data, as described in Table 5. This experiment consists in counting the average number of Sbox evaluations as well as the number of times we abort to try a candidate key, i.e. the number of false positives, until we recover the right key. For comparison, we provide the corresponding quantities in the case of a traditional cryptanalysis-that is, without applying the state-test technique. We have therefore been able to verify that the state-test technique considerably improves the time complexity of the attacks, as predicted.

Table 5. Comparison of the average number of partial encryptions and the number of candidate keys with and without the state-test technique. The results correspond to an average over 2^3 randomly generated keys.

# Pairs of data	With the state-test technique		Without the state-test technique	
	Sbox evaluations	candidate keys	Sbox evaluations	candidate keys
2^4	183.50	14.125	9096.375	216.25
2^5	516.50	5.125	19328.125	73.875
2^6	626.75	1.25	17311.00	9.375
2^7	835.25	1.00	18381.125	7.625
2^8	1278.75	1.00	20942.125	5.50

Fig. 4. Reduced version of CLEFIA used to verify the correctness of the state-test technique. The number 2 on the rightmost word of the ciphertext means that at least two nibbles have a non-zero difference.



4 Applications

We start by providing a brief overview of the importance and impact of each of our applications and a comparison with previous attacks. Table 6 provides the techniques and improvements that are applied in each case, the parameters used as well as the attack complexities. The formulas and techniques that we provide allow for a straightforward application on several ciphers. Thanks to our complexity estimates, we manage to improve on many of the previous attacks. As we have seen in Section 3 and as can be seen in the attack against AES-128, these estimations accurately meet the attack complexities.

Table 6. Summary of the details concerning the new applications. B.B. stands for the black-box key-schedule term. Ch. for *Choosing* Δ_{in}, Δ_{out}, c_{in} *and* c_{out} as presented in [10], where the value provided is the gain in bits for the memory requirements. S.T. stands for state-test, and the amount given is the number of bits that have to be fixed in the plaintext, corresponding to a reduction of the number of involved key bits. Mult. stands for multiple differentials and impossible differentials, and the value given is M as described in Section 2.5. ‡ Without FL layers and whitening keys.

	Parameters				Improvements				Complexity			
Algorithm	Δ_{in}	Δ_{out}	c	K	B.B.	Ch.	S.T.	Mult.	ε	Data	Time	Mem.
AES-128	56	32	68	112	yes	no	8	4	6.1	$2^{113.1}$	$2^{113.1} + 2^{105.1}$	$2^{74.1}$
	64	32	68	112	yes	no	no	4	6	2^{105}	$2^{106.88}$	2^{74}
CRYPTON-128	32	64	81.8	112	yes	no	no	$6 \cdot 2^3$	6.7	$2^{114.92}$	$2^{114.92} + 2^{113.7}$	$2^{88.5}$
ARIA-128	48	32	64	80	yes	no	no	no	5.9	$2^{118.9}$	$2^{118.9} + 2^{80.9}$	$2^{69.9}$
	48	32	64	80	yes	no	no	2^9	7	2^{111}	$2^{111} + 2^{82}$	2^{71}
CLEFIA-128	48	48	80	122	yes	5	16	12	5	$2^{114.4}$	$2^{114.4}$	2^{80}
Camellia-256‡	128	56	168	219	yes	9	16	8	7.7	$2^{117.7}$	$2^{117.7} + 2^{115.7}$	$2^{166.7}$
LBlock	48	32	72	73	negl.	7	8	2^6	2.5	$2^{55.5}$	2^{72}	2^{65}

AES-128. We provide a complete description of one of our attacks that entirely matches the complexity estimations. We chose to detail this attack because it involves the application of the state-test technique, the use of multiples, and the consideration of the black-box term. Another application to AES using multiples and taking the black-box term into account, gives a data complexity of 2^{105} CP, a time complexity of $2^{106.88} C_E$ and a memory complexity of 2^{74} words. This new attack provides a previously unknown trade-off, and the complexities are comparable to those of the best attacks on 7-round AES. Two attacks on AES often cited as the best known are given in Table 1. If we compare their results with our second attack, even though our time complexity is slightly higher, the data complexity is the same and our memory complexity is much better. Given

the importance of AES, these new trade-offs, comparable to the best attacks, are interesting.

CRYPTON-128. We consider several multiples and use $n_{in} = 4$, $n_{out} = 4$ and $m_{out} = 6$. As pointed out previously, the multiple impossible differentials that correspond to the same key bits in the extended rounds can be seen as a reduction in the number of bit conditions, which provides a better memory in the overall complexity while giving exactly the same other complexity parameters.

ARIA-128. In this case, the proposed attacks, are far from being the best, still they result in the best impossible differential attacks against this cipher. Still, this application is a very illustrative example because of the many multiples that can be considered, and it provides the perfect scenario for comparing the use of multiples with the use of a single impossible differential. We discuss the advantages, disadvantages, and how far we can go with this type of attack.

Camellia-256 without FL/FL^{-1} layers and whitening keys. We improve on the previous attack from [10] (which covers the highest number of rounds, starting from the first one), by efficiently combining the state-test technique with multiple impossible differentials. Consequently, we are able to consider more fixed bits for the state-test technique. In addition, we take into account the black box term corresponding to the complex key-schedule. This was not done in the previous best cryptanalysis, and therefore we provide the corrected complexity of the full key-recovery attack. The most important parameters of this application are given in Table 1.

CLEFIA-128. We applied the state-test technique, the use of multiples and the correct way of choosing Δ_{out} and c_{out} to CLEFIA-128 in a similar way to our attack on Camellia-256. We check carefully what happens with the key bits, and we can apply the state-test technique when fixing 16 input bits. We also take into account the black box term corresponding to the key schedule. Finally, we obtain the improved and corrected complexities given in Table 1.

LBlock. In this case, we consider the same starting parameters as in [10]. We can improve the 23-round attack on LBlock by applying the state-test technique with 8 bits fixed on the plaintexts, which could not be done without combining it with multiple impossible differentials. While the techniques we used here were already presented in [10], the attack proposed there on LBlock was much worse because the techniques were not applied in combination. The parameters used are given in Table 1.

4.1 AES-128

We give here a detailed description of our attack against AES-128. We do not recall here the specifications of the AES algorithm but refer to the design paper [14]. To ease the description of the attack, we number the bytes of the 4×4 AES state from 0 to 15, where byte 0 is the byte on the top left corner, byte 1 is the one in the second row from the top and in the leftmost column, and so on.

Previous attacks. During the last 15 years, the security of AES-128 has been extensively analyzed. Among the many different types of attacks considered, impossible differential attacks have long led to the best cryptanalysis. Today, the

most successful cryptanalysis of AES-128 is a Meet-In-The-Middle attack [13] reaching 7 out of 10 AES rounds.

In this work we use improved impossible differential attacks to considerably improve not just the previous impossible differential attacks, but also the memory complexity of the best known attacks against 7-round AES-128 [13], maintaining a similar time complexity. We then provide comparable attacks with new, previously unknown trade-offs. To determine the impossible differential providing the best complexity trade-off for the attack, we carried out an exhaustive search for finding 4-round impossible differentials. For all of them, the application of the `MixColumns` of the last round was omitted. For this search we considered two different types of impossible differentials, covering what we believe to be all impossible differentials on 4 rounds.

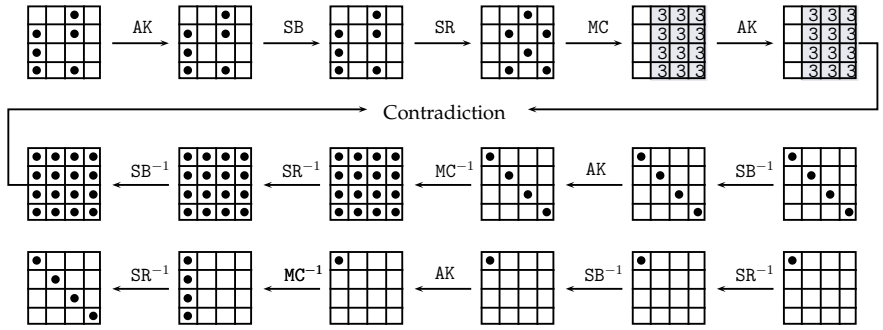
Search of 4-round Impossible Differentials of AES We provide here some details of our automated search of 4-round impossible differentials for AES. To perform our search we considered two types of 4-round impossible differentials. The first type includes differentials where we computed one round in the forward direction and three rounds in the backward direction and applied the miss-in-the-middle technique after the first round. An impossible differential of this type was used in the attack of Mala et al. [29]. The second type, includes differentials with two rounds in the forward and two rounds in the backward direction, with the miss-in-the-middle technique applied after two rounds. An impossible differential of this kind is for example used in the attacks of Bahrak et al. [2], Lu et al. [27] and Zhang et al. [39].

We divided then the impossible differentials found into equivalent classes, where each class contained those differentials where both $\mathcal{D}_X$ and $\mathcal{D}_Y$ had the same active columns and where each of the four columns had the same Hamming weight. The reason for this is that the first operation taking place when expanding the impossible differential backwards and forwards is `MixColumns` (or its inverse). As a consequence, the exact position of the active bytes in one column doesn't alter the attack, only the Hamming weight of each column matters. After this, by taking one representative differential of each class, we checked by an automated program which impossible differential leaded to an attack with the lower possible complexities. For this, we generated all possible differentials from $\mathcal{D}_X$ and $\mathcal{D}_Y$, by taking into account all possibilities after each `MixColumns` operation or its inverse. For each possible attack, we computed the data, memory and time complexities in order to choose the impossible differential that offered the best trade-off among these three quantities.

The conclusion made after this automated search is that the impossible differential providing the best complexity trade-offs for attacking 7 rounds of AES is the one pointed out by Mala et al. in [29]. This impossible differential is such that there are at least three active bytes in the first and the third column of $\mathcal{D}_X$, while the other two columns stay inactive and such that there is exactly one active byte in $\mathcal{D}_Y$. This impossible differential permits on the one hand to take into account in the best way the key schedule of AES-128, rending the number

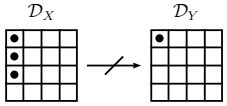
of the key bits that have to be guessed quite reasonable, while on the other hand it permits to minimize the data complexity.

Fig. 5. 4-round impossible differential of AES. A square with a dot symbolizes an active byte, while an empty square stands for inactive bytes. The number 3 on the last three columns after the application of the first `MixColumns` says that at least 3 of the 4 bytes of each column will be active after the application of this operation.



However, we would like to point out that the above impossible differential is not the one leading to the lowest number of key bits that one has to guess. The impossible differential whose $\mathcal{D}_X$ has at least three active bytes in the leftmost column and $\mathcal{D}_Y$ is formed by exactly one active byte can be extended in a way where only 104 bits have to be guessed during the attack, while at least 112 bits are needed with the impossible differential of [29]. However, the induced attack leads to worse data, time and memory complexities than the attack we propose using the differential of [29]. This remark disproves the claim stated in [33] saying that the time complexity of an impossible differential attack only depends on the number of key bits that need to be guessed.

Fig. 6. A 4-round impossible differential of AES, that can require as low as 104 key bits to be guessed during an attack against 7-round AES-128.



Parameters of the basic attack. Since our attack is based on the best previous impossible differential attack of Mala et al. [29], we recall here some of its characteristics.

Their attack is based on several impossible differential paths on $r_{\mathcal{D}} = 4$ rounds. These differ in the pattern of $\mathcal{D}_X$, which can take 4 different forms, each having 3 active bytes in columns 1 and 3, but in different positions. One of these

patterns is represented in Figure 7. Its inactive bytes of columns 1 and 3 are in the positions 0 and 10, but other possibilities are 1 and 11, 2 and 8, and finally 3 and 9. The differentials used in [29] are represented in black in Figure 7. According to our notations, the parameters of the attack described in [29] are: $\Delta_{in} = 64, \Delta_{out} = 32, c_{in} = 46, c_{out} = 22, k_{in} = 80, k_{out} = 32$.

Now, we detail how the application of our techniques leads to a reduced time and memory complexity compared to the Mala et al. attack. First, notice that to conduct this attack, we need to guess four bytes in Round 7 (K_7^0, K_7^7, K_7^{10} and K_7^{13}), while also the following 12 bytes of the first two rounds: K_0^0, K_0^2, K_0^5, K_0^7, K_0^8, K_0^{10}, K_0^{13}, K_0^{15}, K_1^0, K_1^2, K_1^8 and K_1^{10}. However, a study of the AES-128 key schedule reveals that the value of K_1^0 (resp. K_1^2) can be directly computed from the values of K_0^0 and K_0^{13} (resp. of K_0^2 and K_0^{15}), explaining thus why k_{in} is only 80 and not 96.

When applying the generic formulas (3) and (2), with the above parameters, we obtain $N = 2^{68+\varepsilon}$ and $C_N = 2^{101+\varepsilon}$, where ε is a crucial variable that appears in particular in the expression of the probability P of keeping an incorrect key as a candidate: $P \approx 2^{-1.442 \cdot 2^{\varepsilon}}$. Note that different values of ε lead to different time/data/memory trade-offs. Since the key schedule of 7-round AES-128 is non-linear and has a relatively good diffusion after several rounds, we treat it as a black box and then add the potentially expensive term (6) discussed in Section 2.2 to the final time complexity of the attack.

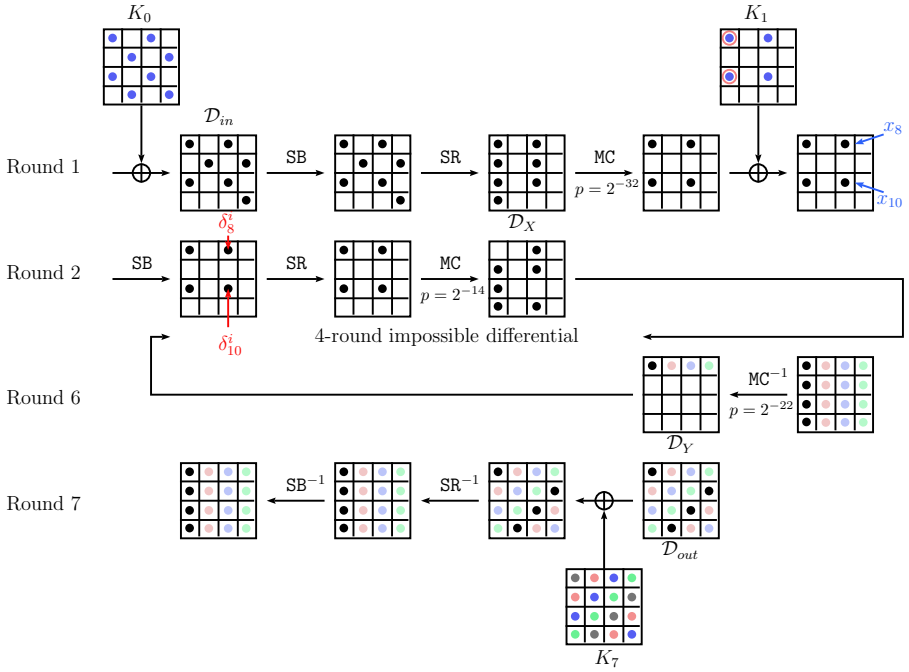
Combining the State-Test Technique with Multiple Differentials The first change we introduce to the attack of [29] is to consider several 4-round impossible differentials that differ in the pattern of $\mathcal{D}_Y$, resulting in four output differences in Round 7, each corresponding to a different anti-diagonal. The involved bits of K_7 form a partition of K_7, as depicted in Figure 7. Our second improvement is the use of the state-test technique. In order to apply the state-test technique, we have to slightly modify the differential of the first rounds used in [29] in order to render one of the previously active bytes of $\mathcal{D}_{in}$ inactive (namely byte 7). We provide a detailed explanation of this in the description of Step 3 of our attack.

To enhance the complexities of our attack, we use the early abort technique together with two precomputed tables:

Table T_1: This table contains all the possible values for the differences lying in the main diagonal of the state after the first `SubBytes` operation. To compute these values, we start from the 2^{16} possible differences of the third column of the state after the `MixColumns` layer in Round 1, and invert the two linear operations `MixColumns` and `ShiftRows`.

Table T_2: Following the same reasoning, we compute the possible values for the differences in bytes $2, 8$ and 13 of the internal state after the first `SubBytes` operation. Contrary to the previous case, we have an additional condition. Indeed, byte 11 of the state outputting the `ShiftRows` layer is inactive, meaning that only 2^8 differences are possible.

Fig. 7. Impossible differential cryptanalysis of 7-round AES-128. The two circled bytes of subkey K_1 come for free by exploiting the key schedule relations. The four colours used for Rounds 6 and 7 correspond to the four output multiple differentials considered.



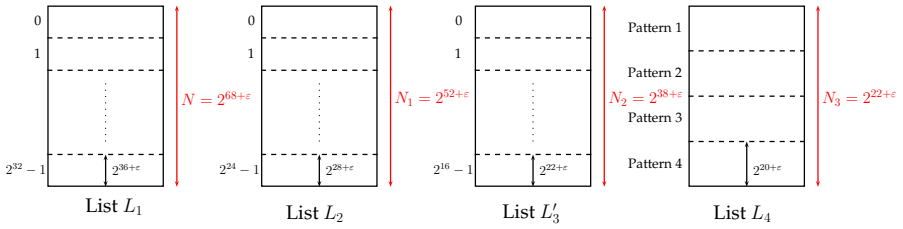
The precomputed tables require a total memory space of $2^{16} + 2^8$ words, which is negligible in comparison to the memory used to store the N pairs.

We now describe the online part of our attack.

Step 1. This step consists in guessing the 32 key bits corresponding to the first diagonal of K_0 (i.e. $K_0^0, K_0^5, K_0^{10}, K_0^{15}$). Starting from $C_N = 2^{107+\varepsilon}$ plaintexts, we extract the $2^{68+\varepsilon}$ pairs that meet the input difference $\mathcal{D}_{in}$ and one of the four possible output differences $\mathcal{D}_{out}$ and store them in a list L_1. We sort this list according to the value of the plaintext difference in the 32-bit diagonal (bytes $0, 5, 10$ and 15), creating then 2^{32} sublists of $2^{36+\varepsilon}$ pairs. We then realize a first guess on the 32 bits of the first diagonal of K_0 (i.e. $K_0^0, K_0^5, K_0^{10}, K_0^{15}$) and follow the next process for each sublist. First, we confront the fixed diagonal plaintext difference with the 2^{16} possible output differences of *Table* T_1, and we use the difference distribution table (DDT) of the Sbox to check if the transitions are possible. If they are, we derive the possible values entering the Sboxes. Due to a well known property of invertible Sboxes, there is one value derived in average for each transition, so we expect 2^{16} values for each sublist. We then combine these values with the previous 32-bit guess on K_0 to deduce the corresponding value of the plaintext diagonal. After that, we look into L_1 and remove the sublists corresponding to plaintexts that have a diagonal value different from

the ones compiled. Out of the 2^{32} possible diagonal values, only 2^{16} are kept, so a proportion of 2^{-16} pairs remains. Note that this sieve corresponds to the probability of having two non-active bytes at the leftmost column after the application of `MixColumns` in the first round.

Fig. 8. The four lists L_1, L_2, L'_3 and L_4 that will be created during the attack. The size of each list as well as the way they are sorted can be visualized.



At this point there are $N_1 = 2^{68-16+\varepsilon} = 2^{52+\varepsilon}$ remaining pairs that we store in a list named L_2 sorted by the difference in the bytes $2, 8$ and 13. Each one of the 2^{24} differences indexes a sublist of $2^{28+\varepsilon}$ pairs.

Step 2. This step is very similar to Step 1, and consists in guessing the bytes K_0^2, K_0^8, K_0^{13} of the subkey K_0. We study each of the 2^{24} sublists together with the 2^8 differences contained in table T_2 to deduce possible values for the inputs of the active Sboxes. As explained before, there will be one such value in average. We then realize a guess of the corresponding bytes of K_0 (K_0^2, K_0^8, K_0^{13}) and deduce by XOR the possible values for the related bytes of the plaintext. The plaintexts of the sublist that are different from those 2^8 candidates are eliminated, resulting in a 2^{-16} sieve. After this step, the number of remaining pairs is $N_2 = 2^{52-16+\varepsilon} = 2^{36+\varepsilon}$. Once again this filter corresponds to the probability to have two non-active bytes at the third column after the application of the `MixColumns` operation of the first round. The complexity up to here is $2^{32+24+36+\varepsilon} = 2^{92+\varepsilon}$ lookup tables. We can compute now for free the values of K_1^0 and K_1^2 from the guesses already realized on K_0.

Step 3. We then repeat the following procedure for each of the $2^{36+\varepsilon}$ pairs left. Starting from the known values of byte 0 and 2 outputting the `MixColumns` operation of Round 1 and of the two subkey bytes of K_1, we compute the values of the two corresponding bytes after the `SubBytes` operation of Round 2. After the application of `ShiftRows`, those two bytes are not in the same column anymore, but are in places 0 and 10. The first column contains then two active bytes, including one which is unknown in position 2, so there are 2^8 possible values for the difference of this column. However, since we know that after passing through `MixColumns` the difference should follow a specific pattern with only three active bytes, the number of possibilities for the unknown byte is restricted. Indeed, if the position of the inactive byte is fixed in $\mathcal{D}_X$, only one possibility remains. However, since here we consider four possible patterns, there are four

possibilities, that we denote by δ_{10}^i, $i = 0, \dots 3$, each one corresponding to a non-active position. The same reasoning holds for the difference in the third column of the state outputting the `ShiftRows` operation of Round 2, in which the known byte difference is in position 10 and the unknown one is byte 8. We denote by δ_8^i, $i = 0, \dots, 3$ the four possibilities for this last one. Since the pattern of the first column leads to a unique possibility for the third column, we have only four possible values for δ_{10}^i and δ_8^i given fixed differences in bytes 0 and 2 at the output of the `SubBytes` operation. Since we know the difference transitions of the active Sboxes in positions 8 and 10 of Round 2, we can refer to the DDT to obtain the values that permit these transitions. Once again there is on average one value for each transition, which we denote, according to their positions, by x_8 and x_{10} (see Figure 7). The new list obtained, named L_3' is of size $4 \cdot 2^{36+\varepsilon} = 2^{38+\varepsilon}$.

The next natural step for continuing the attack is to confront those values with the ones obtained from the plaintext. Indeed, the expression of byte 8 at the entry of `SubBytes` of round 2 is $2S(P^8 + K_0^8) + 3S(P^{13} + K_0^{13}) + S(P^2 + K_0^2) + S(P^7 + K_0^7) + K_1^8$, and the one of byte 10 is $S(P^8 + K_0^8) + S(P^{13} + K_0^{13}) + 2S(P^2 + K_0^2) + 3S(P^7 + K_0^7) + K_1^{10}$, where S is the AES Sbox and where the multiplication is realized in $GF(2^8)$. So to compare it with the four values obtained previously we would need additional key guesses of the values of K_0^7, K_1^8 and K_1^{10}. However, instead of guessing those 3 bytes, we use the state-test technique which allows to decrease the time complexity by a factor of 2^8. The general idea behind this is to study together the pairs that lead to the same values of $S(P^7 + K_0^7) + K_1^8$ and of $3S(P^7 + K_0^7) + K_1^{10}$. To do so, we first compute the known values $2S(P^8 + K_0^8) + 3S(P^{13} + K_0^{13}) + S(P^2 + K_0^2)$ and $S(P^8 + K_0^8) + S(P^{13} + K_0^{13}) + 2S(P^2 + K_0^2)$ and then XOR them respectively to the 4 possible values of x_8 and x_{10}. These two quantities, that we denote by z_1 and z_2 are equal to $z_1 = S(P^7 + K_0^7) + K_1^8$ and $z_2 = 3S(P^7 + K_0^7) + K_1^{10}$. We use the couple (z_1, z_2) to sort the list L_3' into 2^{16} sublists of $2^{38-16+\varepsilon}$ elements. We continue the attack with the sublist L_4 of size $N_3 = 2^{22+\varepsilon}$ pairs. The complexity up to this point is of $2^{32+24+16+22+\varepsilon} = 2^{94+\varepsilon}$ simple operations.

Step 4. In this final step we study the last round of the differential attack. To do so, we divide L_4 into 4 sublists of size $2^{20+\varepsilon}$, each one corresponding to a fixed output pattern. This list is sorted first by $\mathcal{D}_{out}$ and then by value. We then perform the last guess of the attack on the 32 bits of K_7, and check whether the impossible differential is satisfied. If none of the pairs satisfy it, then the partially guessed key bits are returned as a candidate value for the secret subkey bits. We keep four lists of independent possible values for each of the 32 output key bits, each of size $2^{32} \cdot 2^{-1.442 \cdot 2^{\varepsilon-2}}$. This quantity depends on $(\varepsilon - 2)$ instead of ε since we have lists that are four times smaller. The time complexity up to the creation of these lists is $2^{32+24+16+32+\varepsilon} = 2^{104+\varepsilon}$ memory accesses.

An important aspect of the attack is to obtain lists that are small enough that the cost of merging them is not higher than the cost that we have paid so far. This issue arises because we have no way of knowing if the guessed input key bits and the guessed output key bits form a match unless we complete the remaining part of the state (see Section 2.2). The cost of merging these four lists is given directly

from the equation in 2.5: $2^{-1.442 \cdot 2^{\varepsilon}} 2^{4(72+32)} 2^{-3 \cdot 72} \cdot C_{KS} = 2^{-1.442 \cdot 2^{\varepsilon}+72+128}$, where C_{KS} is the cost of the application of the key schedule, and the number of candidates that will remain is the previous term multiplied by $2^{-k_{in}}$ and by $1/C_{KS} = 2^{128-1.442 \cdot 2^{\varepsilon}}$.

Computing C'_E. We estimate C'_E by following the common practice of counting the number of Sbox applications computed in the bottleneck part of the attack (the penultimate of the previous procedure) compared to the number of Sbox applications in the full cipher (as done for instance in [7]). Our computations give $C'_E = 2^{-5.12}$, since we are comparing four Sbox applications to the total of $16 \cdot 7 + 28 = 140$ Sbox applications used in 7 rounds of AES. Finally, we deduce that $C_{KS}/C_E = 2^{-3.6}$.

This attack has data complexity $C_N = 2^{107+\varepsilon}$ CP, time complexity

$$2^{104+\varepsilon} \cdot 2^{-5} + 2^{-1.442 \cdot 2^{\varepsilon}+72+128} \cdot 2^{-3.6} + 2^{128} \cdot 2^{-1.442 \cdot 2^{\varepsilon}} C_E,$$

and memory complexity $N = 2^{68+\varepsilon}$ words. The best time complexity is obtained by taking $\varepsilon = 6.1$, leading to a data complexity of $2^{113.1}$ CP, a time of $2^{105.1}+2^{113.1}$ C_E and a memory complexity of $2^{74.1}$ words.

4.2 CRYPTON-128

This example aims at showing the application of multiple differentials, combined with multiple impossible differentials in impossible differential cryptanalysis.

CRYPTON is an involutive 128-bit block cipher designed by Lim [24] that was a candidate of the AES competition. This block cipher can be parametrized by a key of 128, 192 or 256 bits. The number of rounds is fixed to 12.

Similarly to AES, an internal state of CRYPTON can be seen as a 4×4-byte matrix. Each round is composed by the following four operations:

- γ: a non-linear operation, that uses 8×8-bit involutive Sboxes applied in parallel on the bytes of the internal state.
- π: a linear byte-wise transformation, that executes a 4×4-byte matrix, with branch number 4, on each column of the state.
- τ: a byte transposition of columns into rows with respect to the anti-diagonal of the internal state.
- σ: a byte-wise key addition, identical to the `AddRoundKey` operation of AES.

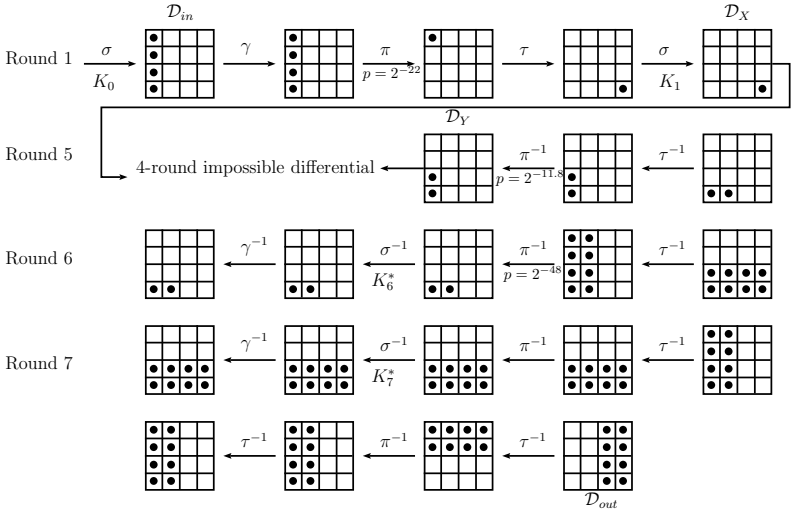
It must be noted that the encryption process of CRYPTON starts by applying σ with the first subkey. Finally, after performing the 12 rounds, the output transformation $\tau \circ \pi \circ \gamma$, i.e. an actual round without the key addition step σ, is applied to the state.

Previous cryptanalysis and our contributions The best previous impossible differential attack against CRYPTON-128 is an impossible differential attack published by Mala et al. [30]. This 7-round cryptanalysis, has a data complexity of 2^{121} CP, a time complexity of $2^{116.2} C_E$ and a memory complexity of 2^{119} words.

Description of the attack In this section we show how to improve all complexity parameters of this attack. For doing this, we jointly use the techniques of multiple differentials and of multiple impossible differentials, for both the first and the last appended rounds. More precisely, we exploit input and output differentials of two types, namely differentials having different $\mathcal{D}_X, \mathcal{D}_Y$, leading to different impossible differentials, while also multiple differentials having a different $\mathcal{D}_{in}, \mathcal{D}_{out}$, as discussed in Sections 2.3 and 2.4. In the same way as for AES, we performed an exhaustive analysis of all 4-round impossible differentials for CRYPTON. We analyzed in an automated way all such impossible differentials, up to equivalent classes, to find out the one that led to the optimal complexities for an attack. In this way, we were able to confirm, that the type of impossible differentials used in [30] is the best choice. An impossible differential of this kind has a single active byte in $\mathcal{D}_X$ and exactly two active bytes in a single column of $\mathcal{D}_Y$ and can be visualized in Figure 9.

This impossible differential covering the rounds 2-5, is then extended one round backwards and two rounds forwards, in exactly the same way as done in [30]. One such extension can be visualized in Figure 9. We skip here most of the details of the attack, as all basic parameters are identical to those of [30]. Instead, we provide details of the multiple (impossible) differentials used. We remind here that we call *multiples* all the differentials that correspond either to multiple differentials or to multiple impossible differentials.

Fig. 9. Impossible differential attack against CRYPTON-128



Input multiples. We use in total 4 input multiples, as this can be seen in Figure 10. These four differentials correspond to $m_{in} = 4$ different $\mathcal{D}_{in}$, described by a different single active column. Each $\mathcal{D}_{in}$ corresponds to one $\mathcal{D}_X$, each one

composed of a single active byte in the last column of the state. However, it would have been possible to take into account four times more $\mathcal{D}_X$ that what we actually use, by considering further any other column with a single active byte. This is depicted in Figure 9 by a $\times 4$ symbol. Nevertheless, for the sake of simplicity and for being in line with previous analyses, the multiple impossible differentials whose conditions do not depend on any key, will be instead taken into account by decreasing c_{in}. In this application, this $\times 4$ parameter is counted in the probability of passing the application π of the first round, that we take to be $p = 2^{-22}$ instead of $p = 2^{-24}$, as can be seen in Figure 9. Note however, that both approaches are equivalent.

Output multiples. We consider here $n_{out} = 4$ differences $\mathcal{D}_Y$. Each $\mathcal{D}_Y$ corresponds to one column with the lower two bytes active. Each $\mathcal{D}_Y$ gives us then $\binom{4}{2} = 6$ possibilities after the application of π in Round 5 for choosing two active bytes within it, leading to $m_{out} = 6$ differences $\mathcal{D}_{out}$. We come finally with the number of $n_{out} \times m_{out} = 4 \times 6 = 24$ output multiples in total. These differentials can be visualized in Figure 10. As explained for the case of input multiples, we could have alternatively considered 4×6 differences $\mathcal{D}_Y$, by taking also into account the six possible positions for the two active bytes. However, by following the same approach as before, we integrate this $\times 6$ factor by decreasing instead c_{out} by a factor of $\log_2 6$.

The remaining parameters of the attack (see Figure 9) are $\Delta_{in} = 32, \Delta_{out} = 64, c_{in} = 24 - \log_2 4 = 22, c_{out} = 14.38 - \log_2 6 + 48 = 59.8, k_A = 32, k_B = 80$.

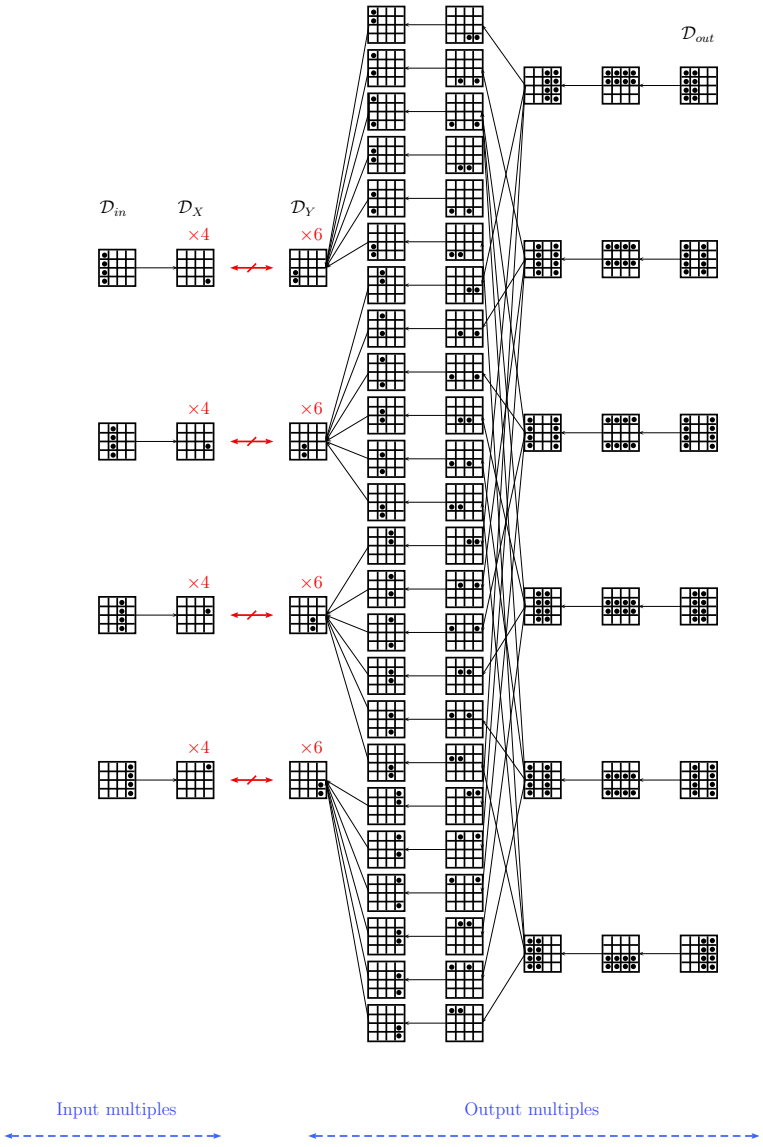
Therefore, by using the formula (5), the data complexity is $C_N = 2^{108.22+\varepsilon}$ CP. The memory complexity, given by the number of pairs N, is $2^{c_{in}+c_{out}+\varepsilon} = 2^{81.8+\varepsilon}$. Finally, as the cost of the key schedule, computed similarly to AES is $C_{KS} = 2^{-3.6}$, C'_E is 2^{-5} and in this application $k_A^{inv} = 128$, the time complexity given by the formula (7) is $2^{112+\varepsilon}2^{-5} + 2^{256-1.442 \cdot 2^{\varepsilon}}2^{-3.6} + 2^{128-1.442 \cdot 2^{\varepsilon}}$ C_E. By taking $\varepsilon = 6.7$ we obtain thus a data complexity of $2^{114.92}$ CP, a time complexity of $2^{113.7}$ C_E and a memory complexity of $2^{88.5}$ 128-bit words.

We can see by the above description that we considerably improve all complexity parameters of the previous best impossible differential attack against CRYPTON-128.

5 ARIA-128

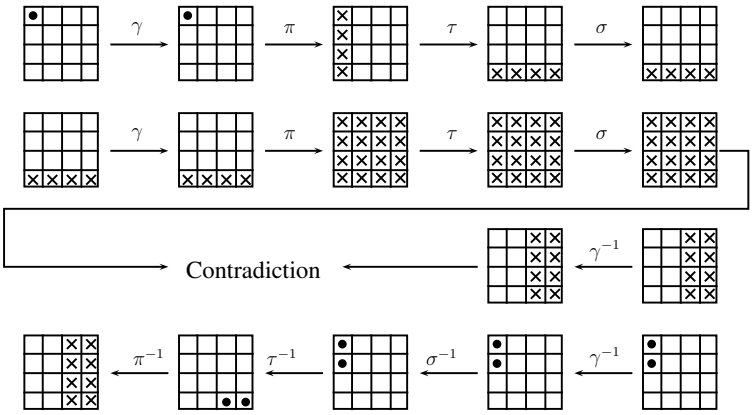
ARIA [19] is a 128-bit block cipher designed in 2003 by Kwon et al. and established as a Korean Standard in 2004. ARIA-128 has 12 rounds and each round is composed of 3 operations. The first operation is the Key Addition (ARK) that simply XORs the 128-bit round key to the state. The second operation is the Substitution Layer (SL) that consists in the parallel application of 4 different Sboxes on every byte of the state. Finally, the Diffusion Layer (DL) is defined by a 16×16 involutory binary matrix ensuring a branch number of 8 and is omitted in the last round. We refer to the design paper [19] for more details.

Fig. 10. Multiples for the attack against CRYPTON-128. The $\times 4$ factor symbolizes that 4 more differences $\mathcal{D}_X$ can be taken into account for each depicted state, by activating another byte in the same row as the one shown. Equally we can consider 6 times more differentials than the ones shown, by choosing different positions for the two active bytes in each column.



Improved 6-round impossible differential attack using *multiples* In this section, we improve on the best impossible differential attack on ARIA-128 [22], that covers up to 6 rounds. This attack is illustrated in Figure 12. We

Fig. 11. 4-round impossible differential of CRYPTON. A square with a dot symbolizes an active byte, while an empty square stands for inactive bytes. A crossed square after the application of π says that at least 3 of the 4 bytes of each column will be active after the application of this operation.

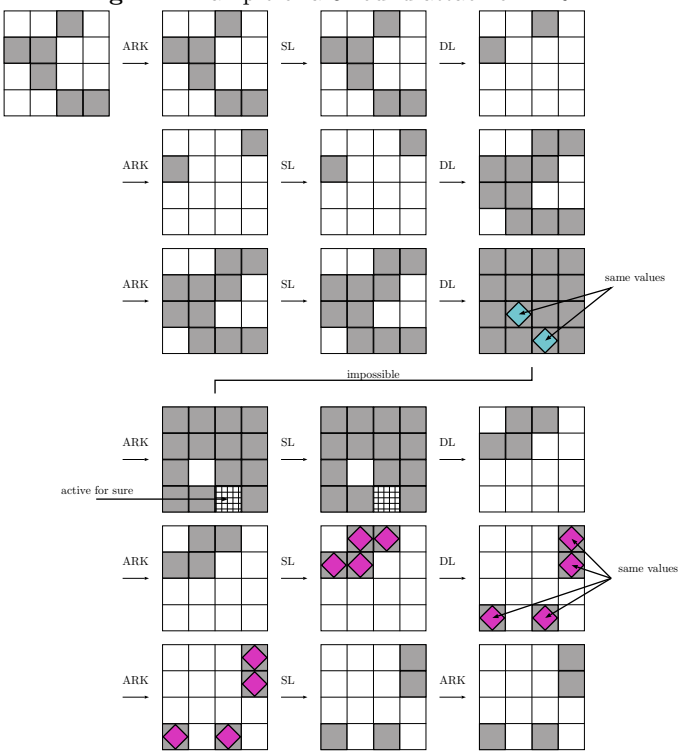


achieve this improvement by using multiple impossible differentials. The main goal of this application is to demonstrate the comparison of a *simple* attack (with only one impossible differential) with a similar attack exploiting multiple differentials instead. We show in particular that in this last case, the value of the variable ε has to be higher, but the data complexity is lower. We consider for our attacks a configuration similar to the one used in [22], i.e. with parameters $\Delta_{in} = 48, \Delta_{out} = 32, c_{in} = 40, c_{out} = 24, k_A = k_{in} = 48, k_B = k_{out} = 32$, as can be seen in Figure 12. We provide now the complexities in the simple and the multiple case.

Simple Case. By directly applying the above attack parameters in the formulas of Section 2, we get a memory complexity of $N = 2^{40+24+\varepsilon_s} = 2^{64+\varepsilon_s}$ words. The data complexity by using Eq. (5) is $C_N = 2^{129+64+\varepsilon_s-48-32} = 2^{113+\varepsilon_s}$ CP. The time complexity can be computed by directly applying Eq. (7) $C_T = 2^{80+\varepsilon_s}2^{-5} + 2^{128+32}2^{-1.442\cdot 2^{\varepsilon_s}}2^{-1.58} + 2^{128}2^{-1.442\cdot 2^{\varepsilon_s}}C_E$. By choosing $\varepsilon_s = 5.9$ the data complexity is $2^{118.9}$ CP , the time complexity is $2^{80.9}C_E$ and the memory complexity is $2^{69.9}$ 128-bit words.

Multiple Case. In order to find the multiple differentials with the same associated parameters as in the simple case described above, we performed an exhaustive search. So we determined how many impossible differentials from two equal active bytes to four equal active bytes exist. We found that there are more than 2^9 such impossible differentials, thus we can consider $M = n_{in} \cdot n_{out} = 2^9$. As we have seen in Section 2.3, the multiple attacks can be seen, except for the data complexity, as applications in parallel of simple attacks, where the associated ε_s are related to the final ε determining the probability of keeping a key as candidate by the following relation: $\varepsilon - \varepsilon_s = \log_2(M)$, which equals 9 in our case. So we have $N = 2^{64+\varepsilon}$ (which is also the memory complexity), and

Fig. 12. Example of a 6-round attack on ARIA



consequently, the data complexity is then: $C_N = 2^{129+64+\varepsilon-48-32-9} = 2^{104+\varepsilon}$ CP. The time complexity can be computed by directly applying the formula for the time complexity and the modification from Section 2.5: $C_T = 2^{80+\varepsilon}2^{-5} + 2^{256}2^{-1.442\cdot 2^{\varepsilon}}2^{-1.58} + 2^{128}2^{-1.442\cdot 2^{\varepsilon}}C_E$. By taking $\varepsilon = 7$ we get a data complexity of 2^{111} CP, a time complexity of $2^{82}C_E$ and a memory complexity of 2^{71} 128-bit words.

Comparing both. An interesting generic question is: Are there cases where the simple attack might provide a better complexity? As one can see from above, if we wanted to obtain the same data complexity, we should take an ε such that $\varepsilon = \varepsilon_s + 9$. In this case, the memory complexity of the multiple case is a factor of 2^9 times higher than in the simple one. Lets see what happens with the time complexity. The first and last terms of the time complexity are equal. The difference might come from the middle term: $2^{128+32}2^{-1.442\cdot 2^{\varepsilon_s}}2^{-3.6}$ and $2^{256}2^{-1.442\cdot 2^{\varepsilon}}2^{-3.6}$. We see that in the multiple case, we have the simple case term multiplied by $2^{96}\cdot 2^{-1.442\cdot(2^9-1)\cdot 2^{\varepsilon_s}} = 2^{-640.86\cdot 2^{\varepsilon}}$, which provides a better complexity. Despite this, when the bottleneck term of the time complexity for the best attacks is not the second term but the first, as is the case of the results on ARIA, while the data complexity is always much worse in the simple case,

the time complexity might be slightly better, given by the smaller ε_s that we can take into account.

6 Conclusion

In this paper we presented new techniques for improving impossible differential attacks. Furthermore, we showed that the nature of the key schedule has a non-negligible impact on the time complexity of such attacks, and provided a new complexity formula taking this phenomenon into account. We applied these new techniques, individually and in combination to various ciphers, based on both SPN and Feistel constructions. From this point of view, our work complements the results of [10] where only Feistel ciphers were analyzed. We showed here that our techniques, as well as those introduced in [10], work on both constructions. However, there are small differences in the extent of the applicability of these techniques. For example, we noticed that applying multiple differentials in impossible differential cryptanalysis is somewhat easier on SPN ciphers, because linear layers of `MixColumns` type offer more possibilities for extending a differential, hence naturally provide more input/output differences. On the other hand, the state-test technique applies more easily to Feistel ciphers. A natural explanation for this is that in SPN ciphers, even if the state-test technique can almost always be applied, the gain in the complexity generally leads to an equivalent loss in data complexity, because a part of the active part of the plaintexts has to be fixed.

We also compared attacks based on multiple (impossible) differentials with equivalent attacks exploiting only a single differential. We showed that when exploiting multiple differentials, the data complexity is always lower. However, the gain in the time complexity is not always clear, and a simple attack can sometimes lead to a better time complexity.

Additionally, in order to verify and validate the applicability of the proposed techniques, we implemented two of the techniques on toy ciphers. These experiments confirm that our theoretical estimates are indeed good estimates of the complexities. However, we insist that for an exact determination of the complexity, one must perform the detailed attack step by step.

References

1. Aoki, K., Ichikawa, T., Kanda, M., Matsui, M., Moriai, S., Nakajima, J., Tokita, T.: Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms - Design and Analysis. In: Selected Areas in Cryptography - SAC 2000. LNCS, vol. 2012, pp. 39–56. Springer (2000)
2. Bahrak, B., Aref, M.R.: Impossible Differential Attack on Seven-Round AES-128. IET Information Security 2(2), 28–32 (2008)
3. Biham, E., Biryukov, A., Shamir, A.: Cryptanalysis of Skipjack Reduced to 31 Rounds Using Impossible Differentials. In: EUROCRYPT'99. LNCS, vol. 1592, pp. 12–23. Springer (1999)

4. Biryukov, A., Derbez, P., Perrin, L.: Differential Analysis and Meet-in-the-Middle Attack Against Round-Reduced TWINE. In: Fast Software Encryption - FSE 2015. LNCS, vol. 9054, pp. 3–27. Springer (2015)
5. Blondeau, C.: Impossible differential attack on 13-round Camellia-192. Inf. Process. Lett. 115(9), 660–666 (2015)
6. Blondeau, C., Minier, M.: Analysis of Impossible, Integral and Zero-Correlation Attacks on Type-II Generalized Feistel Networks Using the Matrix Method. In: FSE 2015. LNCS, vol. 9054, pp. 92–113. Springer (2015)
7. Bogdanov, A., Khovratovich, D., Rechberger, C.: Biclique Cryptanalysis of the Full AES. In: ASIACRYPT'11. LNCS, vol. 7073, pp. 344–371. Springer (2011)
8. Bogdanov, A., Knudsen, L.R., Leander, G., Paar, C., Poschmann, A., Robshaw, M.J.B., Seurin, Y., Vikkelsoe, C.: PRESENT: An Ultra-Lightweight Block Cipher. In: CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer (2007)
9. Boura, C., Minier, M., Naya-Plasencia, M., Suder, V.: Improved Impossible Differential Attacks against Round-Reduced LBlock. IACR Cryptology ePrint Archive 2014, 279 (2014), http://eprint.iacr.org/2014/279
10. Boura, C., Naya-Plasencia, M., Suder, V.: Scrutinizing and Improving Impossible Differential Attacks: Applications to CLEFIA, Camellia, LBlock and Simon. In: ASIACRYPT'14 (I). LNCS, vol. 8873, pp. 179–199. Springer (2014)
11. Derbez, P.: Note on Impossible Differential Attacks. In: Fast Software Encryption - FSE 2016. LNCS, vol. 9783, pp. 416–427. Springer (2016)
12. Derbez, P., Fouque, P.A.: Exhausting Demirci-Selçuk Meet-in-the-Middle Attacks Against Reduced-Round AES. In: Fast Software Encryption - FSE 2013. LNCS, vol. 8424, pp. 541–560. Springer (2013)
13. Derbez, P., Fouque, P.A., Jean, J.: Improved Key Recovery Attacks on Reduced-Round AES in the Single-Key Setting. In: EUROCRYPT'13. LNCS, vol. 7881, pp. 371–387. Springer (2013)
14. FIPS 197: Announcing the Advanced Encryption Standard (AES). National Institute for Standards and Technology, Gaithersburg, MD, USA (November 2001)
15. Guo, J., Peyrin, T., Poschmann, A., Robshaw, M.J.B.: The LED Block Cipher. In: CHES 2011. LNCS, vol. 6917, pp. 326–341. Springer (2011)
16. Jean, J., Naya-Plasencia, M., Peyrin, T.: Multiple Limited-Birthday Distinguishers and Applications. In: SAC'13. LNCS, vol. 8282. Springer (2014)
17. Kim, J., Hong, S., Lee, S., Song, J.H., Yang, H.: Truncated Differential Attacks on 8-Round CRYPTON. In: ICISC 2003. Lecture Notes in Computer Science, vol. 2971, pp. 446–456. Springer (2004)
18. Knudsen, L.R.: DEAL – A 128-bit cipher. Technical Report, Department of Informatics, University of Bergen, Norway (1998)
19. Kwon, D., Kim, J., Park, S., Sung, S.H., Sohn, Y., Song, J.H., Yeom, Y., Yoon, E., Lee, S., Lee, J., Chee, S., Han, D., Hong, J.: New Block Cipher: ARIA. In: ICISC 2003. LNCS, vol. 2971, pp. 432–445. Springer (2004)
20. Li, L., Jia, K., Wang, X., Dong, X.: Meet-in-the-Middle Technique for Truncated Differential and Its Applications to CLEFIA and Camellia. In: Fast Software Encryption - FSE 2015. LNCS, vol. 9054, pp. 48–70. Springer (2015)
21. Li, R., Sun, B., Zhang, P., Li, C.: New Impossible Differential Cryptanalysis of ARIA. Cryptology ePrint Archive, Report 2008/227 (2008)
22. Li, S., Song, C.: Improved Impossible Differential Cryptanalysis of ARIA. In: ISA 2008. pp. 129–132 (2008)
23. Li, X., Jin, C., Fu, F.: Improved Results of Impossible Differential Cryptanalysis on Reduced FOX. Comput. J. 59(4), 541–548 (2016)

24. Lim, C.H.: A Revised Version of Crypton - Crypton V1.0. In: FSE'99. LNCS, vol. 1636, pp. 31–45. Springer (1999)
25. Liu, Y., Li, L., Gu, D., Wang, X., Liu, Z., Chen, J., Li, W.: New Observations on Impossible Differential Cryptanalysis of Reduced-Round Camellia. In: FSE'12. LNCS, vol. 7549, pp. 90–109. Springer (2012)
26. Liu, Z., Gu, D., Liu, Y., Li, J., Lei, W.: Linear cryptanalysis of aria block cipher. In: Information and Communications Security, LNCS, vol. 7043, pp. 242–254. Springer (2011)
27. Lu, J., Dunkelman, O., Keller, N., Kim, J.: New Impossible Differential Attacks on AES. In: INDOCRYPT'08. LNCS, vol. 5365, pp. 279–293. Springer (2008)
28. Mala, H.: Private communication (2014)
29. Mala, H., Dakhilalian, M., Rijmen, V., Modarres-Hashemi, M.: Improved Impossible Differential Cryptanalysis of 7-Round AES-128. In: INDOCRYPT'10. LNCS, vol. 6498, pp. 282–291. Springer (2010)
30. Mala, H., Shakiba, M., Dakhilalian, M.: New impossible differential attacks on reduced-round Crypton. Computer Standards & Interfaces 32(4), 222–227 (2010)
31. Minier, M., Gilbert, H.: Stochastic Cryptanalysis of Crypton. In: FSE 2000. LNCS, vol. 1978, pp. 121–133. Springer (2001)
32. M.Marine: Improving impossible-differential attacks against Rijndael-160 and Rijndael-224. Designs, Codes and Cryptography pp. 1–13 (2016)
33. Shakiba, M., Dakhilalian, M., Mala, H.: On computational complexity of impossible differential cryptanalysis. Inf. Process. Lett. 114(5), 252–255 (2014)
34. Shirai, T., Shibutani, K., Akishita, T., Moriai, S., Iwata, T.: The 128-Bit Blockcipher CLEFIA (Extended Abstract). In: Fast Software Encryption - FSE 2007. LNCS, vol. 4593, pp. 181–195. Springer (2007)
35. Tsunoo, Y., Tsujihara, E., Shigeri, M., Suzaki, T., Kawabata, T.: Cryptanalysis of CLEFIA using multiple impossible differentials. In: ISITA'08. pp. 1–6 (2008)
36. Wu, W., Zhang, L.: LBlock: A Lightweight Block Cipher. In: ACNS 2011. LNCS, vol. 6715, pp. 327–344. Springer (2011)
37. Wu, W., Zhang, W., Feng, D.: Impossible Differential Cryptanalysis of Reduced-Round ARIA and Camellia. J. Comput. Sci. Technol. 22(3), 449–456 (2007)
38. Yang, Q., Hu, L., Sun, S., L.Song: Related-key Impossible Differential Analysis of Full Khudra. IACR Cryptology ePrint Archive 2015, 840 (2015), http://eprint.iacr.org/2015/840
39. Zhang, W., Wu, W., Feng, D.: New Results on Impossible Differential Cryptanalysis of Reduced AES. In: ICISC'07. LNCS, vol. 4817, pp. 239–250. Springer (2007)